Unit 2

Scan Conversion Algorithm

Scan Conversion

- A procedure used to digitize or rasterize pixel data available on frame buffer.
- The process of representing continuous graphics object as a collection of discrete pixels is called scan conversion. For e.g. a line is defined by its two end points and the line equation.

➢ Scan conversion of a point

- Scan conversion of a point requires the two data that are pixel position and color for display.
- In C, a point be scan converted using function putpixel() defined in library. Header file is
  <graphics.h>
  putpixel(x, y, color)

  Here, x & y represent pixel position on 2D display.

➢ Scan Conversion of line

Say $y = mx + b$ be the equation of line with end point (x_1, y_1) and (x_2, y_2) then,

$m = \frac{y_2 - y_1}{x_2 - x_1}$

$\therefore m = \frac{\Delta y}{\Delta x}$ ………………. (i)

Here, m represents the slope of line path where by Δx & Δy gives the deflection needed towards horizontal and vertical direction to get new pixel from current pixel position.

- Slope of line also describes the nature and characteristics of line that is going to display.

Line Drawing algorithm

a) Digital Differential Analyzer (DDA) algorithm (Incremental algorithm)
b) Bresenham's line drawing algorithm

a) Digital Differential Analyzer (DDA) algorithm

It is a scan conversion line algorithm based on calculating either Δx or Δy using equation $m = \frac{\Delta y}{\Delta x}$

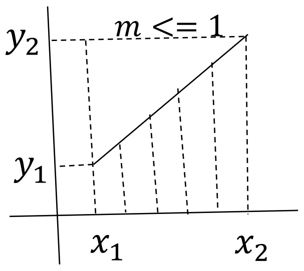
The equation of the line is given as;

$y = mx + b$ …………..(i)

$m = (y_2 - y_1)/(x_2 - x_1)$ ……………. (ii)

For any interval Δx , corresponding interval is given by $\Delta y = m\Delta x$.

If $m <= 1$, and start point is left endpoint then $\Delta x = 1$ and



$$y_{k+1} = y_k + m$$

If $m <= 1$, and start point is right endpoint, then $\Delta x = -1$ and

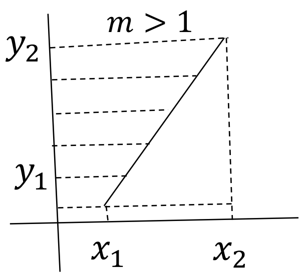
$$y_{k+1} = y_k - m$$

If $m > 1$, and start point is left endpoint, then $\Delta y = 1$ and

$$x_{k+1} = x_k + \frac{1}{m}$$

If $m > 1$, and start point is right endpoint, then $\Delta y = -1$ and

$$x_{k+1} = x_k - \frac{1}{m}$$



Advantages of DDA:

- It is simple to understand.
- It requires no special skills for implementation.
- It is faster method than direct use of the line equation y=mx+c.

Disadvantages of DDA:

- m is stored in floating point number.
- Accumulation of round-off error in successive additions can cause calculated pixel positions to drift away from the actual line path for long line segments.
- Rounding operations and floating-point-arithmetic are time consuming.

Examples

Q. Digitized the line with end points (0, 0) and (4, 5) using DDA.

Solution:

Given,
$(x_1, y_1) = (0, 0)$
$(x_2, y_2) = (4, 5)$
$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5-0}{4-0} = 1.25$

Since, $m > 1$, from DDA algorithm we have;

$y_{k+1} = y_k + 1$

$x_{k+1} = x_k + \frac{1}{m}$

x	y	x-plot	y-plot	(x, y)
0	0	0	0	(0, 0)
0.8	1	1	1	(1, 1)
1.6	2	2	2	(2, 2)
2.4	3	2	3	(2, 3)
3.2	4	3	4	(3, 4)
4	5	4	5	(4, 5)

Q. Digitized the line with end points (3, 7) and (8, 3) using DDA.

Solution:

Given,

$(x_1, y_1) = (3, 7)$

$(x_2, y_2) = (8, 3)$

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-7}{8-3} = -0.8$

Since, m<1, from DDA algorithm we have;

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k + m$

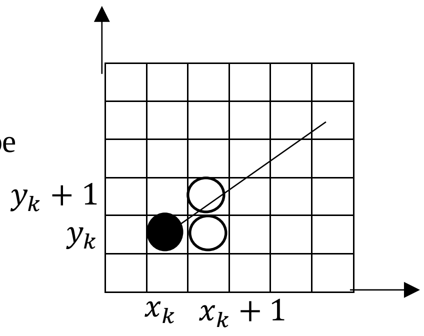
x	y	x-plot	y-plot	(x, y)
3	7	3	7	(3, 7)
4	6.2	4	6	(4, 6)
5	5.4	5	5	(5, 5)
6	4.6	6	5	(6, 5)
7	3.8	7	4	(7, 4)
8	3	8	3	(8, 3)

b) *<u>Bresenham's line drawing algorithm</u>*

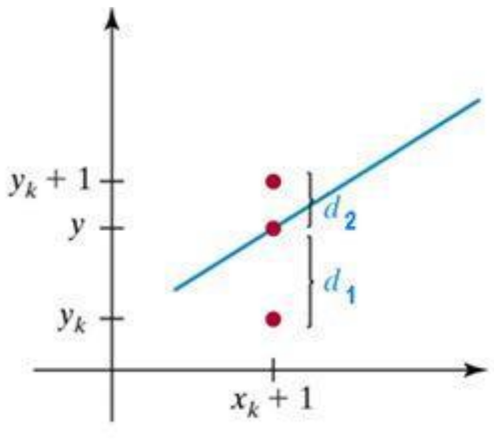
<u>Case I:</u> $0 < m < 1$

Let (x_k, y_k) is pixel at k^{th} step then next point to plot may be either $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$.

Let d_1 & d_2 be the separation of pixel position $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$ from the actual line path.



$$y = mx + b$$



Then at sampling point $(x_k + 1)$

$$y = m(x_k + 1) + b$$

From figure,

$d_1 = y - y_k = m(x_k + 1) + b - y_k$

$d_2 = (y_k + 1) - y = (y_k + 1) - m(x_k + 1) - b$

Now, $d_1 - d_2 = 2m(x_k + 1) - (y_k + 1) - y_k + 2b = 2m(x_k + 1) - 2y_k + 2b - 1$

Let us define a decision parameter P_k for the k^{th} step by

$P_k = \Delta x(d_1 - d_2)$

$\Delta x > 0$ Therefore, $P_k < 0$ if $d_1 < d_2$

$P_k \geq 0$ if $d_1 > d_2$

$\therefore P_k = \Delta x(d_1 - d_2) = \Delta x\{2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1\} = 2\Delta y.x_k - 2\Delta x.y_k + c$ ……(i)

Where the constant c =$2\Delta y + \Delta x(2b - 1)$.

Now, for next step;

$P_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$ ……..(ii)

From (i) & (ii)

$\therefore P_{k+1} - P_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$

$\therefore P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$ [Since, $x_{k+1} = x_k + 1$]

Where, $y_{k+1} - y_k$= 0 or 1

If $P_k < 0$,

$y_{k+1} = y_k$ so $P_{k+1} = P_k + 2\Delta y$

If $P_k \geq 0$,

$y_{k+1} = y_k + 1$ so $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Therefore, initial decision parameter,

$P_0 = 2\Delta y.x_0 - 2\Delta x.y_0 + c$ [from (i)]

$=2\Delta yx_0 - 2\Delta xy_0 + 2\Delta y + \Delta x(2b - 1)$

$=2\Delta yx_0 - 2\Delta xy_0 + 2\Delta y + 2b\Delta x - \Delta x$

$=2\Delta yx_0 - 2\Delta xy_0 + 2\Delta y + 2(y_0 - \frac{\Delta y}{\Delta x}x_0)\Delta x - \Delta x$

$=2\Delta yx_0 - 2\Delta xy_0 + 2\Delta y + 2\Delta xy_0 - 2\Delta yx_0 - \Delta x$

$=2\Delta y - \Delta x$

$$\boxed{P_0 = 2\Delta y - \Delta x}$$

Algorithm

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Load (x_0, y_0) into the frame buffer i.e. plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$ and $2\Delta y$-$2\Delta x$ and obtain the starting value for the decision parameter as
$$P_0 = 2\Delta y - \Delta x$$
4. At each x_k, along the line, starting at k=0, perform the following tests:
If $P_k < 0$, then next point to plot is $(x_k + 1, y_k)$ and
$$P_{k+1} = P_k + 2\Delta y$$
Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and
$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$
5. Repeat step 4 Δx times.

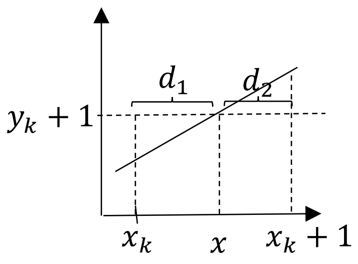
[***Note:*** *For m>1, just interchange the role of x & y*]

Case II: $m > 1$

Let (x_k, y_k) is pixel at k^{th} step then next point to plot may be either $(x_k, y_k + 1)$ or $(x_k + 1, y_k + 1)$.

Let d_1 & d_2 be the separation of pixel position $(x_k, y_k + 1)$ and $(x_k + 1, y_k + 1)$ from the actual line path.

$$y = mx + b$$



The actual value of x is given by;

$$x = (y - b)/m$$

Then at sampling point $(y_k + 1)$

$$x = (y_k + 1 - b)/m$$

From figure,

$d_1 = x - x_k = \frac{y_k+1-b}{m} - x_k = \frac{y_k+1-b-mx_k}{m}$

$d_2 = (x_k + 1) - x = (x_k + 1) - \frac{y_k+1-b}{m} = \frac{mx_k+m-y_k-1+b}{m}$

Now, $d_1 - d_2 = \frac{2y_k-2mx_k-2b-m+2}{m}$

Let us define a decision parameter P_k for the k^{th} step by

$P_k = \Delta y(d_1 - d_2)$

$\Delta y > 0$ Therefore, $P_k < 0$ if $d_1 < d_2$

$P_k \geq 0$ if $d_1 \geq d_2$

$$\therefore\ P_k = \Delta y(d_1 - d_2) = \Delta y\left\{\frac{2y_k - 2\frac{\Delta y}{\Delta x}x_k - 2b - \frac{\Delta y}{\Delta x} + 2}{\frac{\Delta y}{\Delta x}}\right\} = 2\Delta x y_k - 2\Delta y x_k + 2(1-b)\Delta x - \Delta y$$

$\therefore\ P_k = 2\Delta x y_k - 2\Delta y x_k + c$ …….. (i)

Where, $c = 2(1-b)\Delta x - \Delta y$

Now, for next step;

$P_{k+1} = 2\Delta x . y_{k+1} - 2\Delta y . x_{k+1} + c$ ……..(ii)

From (i) & (ii)

$\therefore\ P_{k+1} - P_k = 2\Delta x(y_{k+1} - y_k) - 2\Delta y(x_{k+1} - x_k)$

$\therefore\ P_{k+1} = P_k + 2\Delta x - 2\Delta y(x_{k+1} - x_k)$ [Since, $y_{k+1} = y_k + 1$]

Where, $x_{k+1} - x_k$= 0 or 1

If $P_k < 0$,

$x_{k+1} = x_k$ so $P_{k+1} = P_k + 2\Delta x$

If $P_k \geq 0$,

$x_{k+1} = x_k + 1$ so $P_{k+1} = P_k + 2\Delta x - 2\Delta y$

Therefore, initial decision parameter,

$$\begin{aligned}P_0 &= 2\Delta x y_0 - 2\Delta y x_0 + c\\ &= 2\Delta x y_0 - 2\Delta y x_0 + 2(1-b)\Delta x - \Delta y\\ &= 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2b\Delta x - \Delta y\\ &= 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2(y_0 - m x_0)\Delta x - \Delta y\\ &= 2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x - 2(y_0 - \tfrac{\Delta y}{\Delta x}x_0)\Delta x - \Delta y\end{aligned}$$

$P_0 = 2\Delta x - \Delta y$

Examples

Q. Digitize the line with endpoints (20, 10), (30, 18) using Bresenham's line algorithm.

Solution:

Here,
$(x_1, y_1) = (20, 10)$
$(x_2, y_2) = (30, 18)$
$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18-10}{30-20} = 0.8 < 1$

Here, $\Delta x = 10,\ \Delta y = 8,\ 2\Delta y = 16, 2\Delta y - 2\Delta x = -4$
The initial decision parameter (P_0) $= 2\Delta y - \Delta x = 16 - 10 = 6 > 0$

Since, for the Bresenham's line drawing algorithm of slope $m < 1$, we have

If $P_k \geq 0$,

$$y_{k+1} = y_k + 1 \ \& \ x_{k+1} = x_k + 1 \quad \text{and} \quad P_{k+1} = P_k + 2\Delta y - 2\Delta x$$

If $P_k < 0$,

$$y_{k+1} = y_k \ \& \ x_{k+1} = x_k + 1 \quad \text{and} \quad P_{k+1} = P_k + 2\Delta y$$

k	P_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

***Q. Digitize the line with endpoints (20, 15) & (30, 30) using Bresenham's line algorithm*.**

Solution:

Here,

$(x_1, y_1) = (20, 15)$

$(x_2, y_2) = (30, 30)$

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{30-15}{30-20} = 1.5 > 1$

Here, $\Delta x = 10,\ \Delta y = 15,\ 2\Delta x = 20,\ 2\Delta x - 2\Delta y = -10$

The initial decision parameter (P_0) $= 2\Delta x - \Delta y$ =20-15=5>0

Since, for the Bresenham's line drawing algorithm of slope $m > 1$, we have

If $P_k \geq 0$,

$$x_{k+1} = x_k + 1 \ \& \ y_{k+1} = y_k + 1 \quad \text{and} \quad P_{k+1} = P_k + 2\Delta x - 2\Delta y$$

If $P_k < 0$,

$$x_{k+1} = x_k \ \& \ y_{k+1} = y_k + 1 \quad \text{and} \quad P_{k+1} = P_k + 2\Delta x$$

k	P_k	(x_{k+1}, y_{k+1})
0	5	(21, 16)
1	-5	(21, 17)
2	15	(22, 18)
3	5	(23, 19)
4	-5	(23, 20)
5	15	(24, 21)
6	5	(25, 22)
7	-5	(25, 23)
8	15	(26, 24)
9	5	(27, 25)
10	-5	(27, 26)

11	15	(28, 27)
12	5	(29, 28)
13	-5	(29, 29)
14	15	(30, 30)

Q. How would you digitize a line with end points A(6, 12) and B(10, 5) using Bresenham's line drawing algorithm?

Solution:

Here,

$(x_1, y_1) = (6, 12)$

$(x_2, y_2) = (10, 5)$

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5-12}{10-6} = -1.75$

$\therefore |m| = 1.75 > 1$

Here, $\Delta x = |x_2 - x_1| = 4,\ \ \Delta y = |y_2 - y_1| = 7,\ \ 2\Delta x = 20, 2\Delta x - 2\Delta y = -10$

The initial decision parameter (P_0) $= 2\Delta x - \Delta y = 8 - 7 = 1 > 0$

Since, for the Bresenham's line drawing algorithm of slope $m > 1$, we have

(Here, $y_1 > y_2$ so y is decremented in successive step)

If $P_k \geq 0$,

$$x_{k+1} = x_k + 1\ \ \&\ y_{k+1} = y_k - 1\ \ \text{and}\ \ P_{k+1} = P_k + 2\Delta x - 2\Delta y$$

If $P_k < 0$,

$$x_{k+1} = x_k\ \&\ y_{k+1} = y_k - 1\ \ \text{and}\ \ P_{k+1} = P_k + 2\Delta x$$

k	P_k	(x_{k+1}, y_{k+1})
0	1	(7, 11)
1	-5	(7, 10)
2	3	(8, 9)
3	-3	(8, 8)
4	5	(9, 7)
5	-1	(9, 6)
6	7	(10, 5)

Advantages of Bresenham's line algorithm (BLA) over DDA:

- In DDA algorithm each successive point is computed in floating point, so it required more time and memory space. While in BLA each successive point is calculated in integer value or whole number. So it requires less time and less memory
- In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it does not need to round, so there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while in BLA line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse, and other curves.

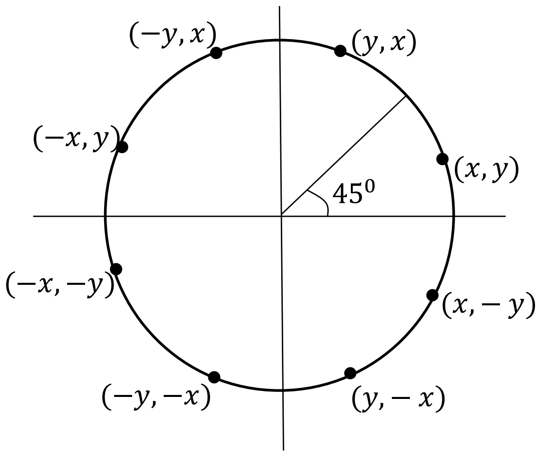
➢ Circle

Circle is defined as a set of points that are all at a given distance ‘r’ from the center position (x_c, y_c). Equation of circle centered at (x_c, y_c) with radius ‘r’ is

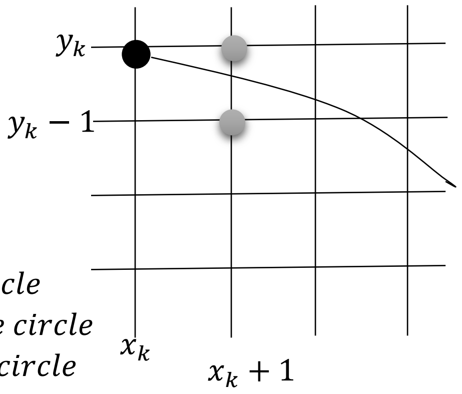
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Symmetry of Circle

Calculation of circle point (x, y) in one octant yields the circle points shown for the other seven octants.



➢ *Midpoint Circle Algorithm*



Assume that we have just plotted point *(xₖ, yₖ)*.
The next point is a choice between *(xₖ+1, yₖ)* and *(xₖ+1, yₖ-1)*. We would like to choose the point that is nearest to the actual circle.
Let us define a circle function as;

$$f(x,y) = x^2 + y^2 - r^2 = \begin{cases} 0 & \textit{if } (x,y) \textit{ is on the circle} \\ > 0 & \textit{if } (x,y) \textit{ is outside the circle} \\ < 0 & \textit{if } (x,y) \textit{ is inside the circle} \end{cases}$$

By evaluating this function at the midpoint between the candidate pixels we can make our decision.
Our decision variable can be defined as:

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

If $p_k < 0$ the midpoint is inside the circle and and the pixel at y_k is closer to the circle Otherwise, $y_k - 1$ is closer.

The decision parameter for next position is at $x_{k+1} + 1$ i.e $x_k + 2$.

$$p_{k+1} = f\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right) = (x_k + 2)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

Now,

$$p_{k+1} - p_k = (x_k + 2)^2 + \left(y_{k+1} - \tfrac{1}{2}\right)^2 - r^2 - (x_k + 1)^2 - \left(y_k - \tfrac{1}{2}\right)^2 + r^2$$

$$p_{k+1} = p_k + x_k^2 + 4x_k + 4 + y_{k+1}^2 - y_{k+1} + \tfrac{1}{4} - x_k^2 - 2x_k - 1 - y_k^2 + y_k - \tfrac{1}{4}$$

$p_{k+1} = p_k + 2x_k + 3 + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$
$\quad = p_k + 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$
Where, y_{k+1} is either y_k or y_{k-1} depending on the sign of p_k.
If $p_k < 0$;
$$y_{k+1} = y_k \Longrightarrow p_{k+1} = p_k + 2x_{k+1} + 1$$
If $p_k \geq 0$;
$$y_{k+1} = y_k - 1 \Longrightarrow p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
For initial decision parameter;
$(x_0, y_0) = (0, r)$
$p_0 = f\left(1, r - \frac{1}{2}\right) = 1 + (r - \frac{1}{2})^2 - r^2 = \frac{5}{4} - r$
All increments are integer, rounding $\frac{5}{4}$ will give 1 so,
$$p_0 = 1 - r$$

Algorithm

1. Input radius 'r' and circle center (x_c, y_c) and obtain the first point on the circumference of a circle centered on origin as
$$(x_0, y_0) = (0, r)$$
2. Calculate the initial value of the decision parameter as
$$p_0 = \frac{5}{4} - r$$
3. At each x_k position, starting at k=0, perform the following test:
If $p_k < 0$, the next point on circle is $(x_k + 1, y_k)$ and
$$p_{k+1} = p_k + 2x_{k+1} + 1$$
Otherwise, the next point on circle is $(x_k + 1, y_k - 1)$ and
$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
4. Determine the symmetry point in other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the co-ordinate values,
$$x = x + x_c, \qquad y = y + y_c$$
6. Repeat step 3 through 5 until $x \geq y$.

Examples

Q. Digitize the circle $x^2 + y^2 = 100$ in first octant.
Solution:
Here,
Center = (0, 0)
Radius (r) =10
Initial point = (0, r) = (0, 10)
Initial decision parameter $p_0 = 1 - r = 1 - 10 = -9$
From mid-point circle algorithm we have;
If $p_k < 0$;
Plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$
$p_k \geq 0$;
Plot $(x_k + 1, y_k - 1)$ and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

Q. Digitize the circle with radius r =10 centered (3, 4) in first octant.

Solution:

Note: When center is not origin, we first calculate the octants points of the circle in the same way as the center at origin then add the given circle center on each calculated pixel.

Here,
Center = (3, 4)
Radius (r) =10
Initial point = (0, r) = (0, 10)
Initial decision parameter $p_0 = 1 - r = 1 - 10 = -9$
From mid-point circle algorithm we have;
If $p_k < 0$;
Plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$
$p_k \geq 0$;
Plot $(x_k + 1, y_k - 1)$ and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

k	p_k	(x_{k+1}, y_{k+1}) at (0, 0)	$2x_{k+1}$	$2y_{k+1}$	(x_{k+1}, y_{k+1}) at (3, 4)
0	-9	(1, 10)	2	20	(4, 14)
1	-6	(2, 10)	4	20	(5, 14)
2	-1	(3, 10)	6	20	(6, 14)
3	6	(4, 9)	8	18	(7, 13)
4	-3	(5, 9)	10	18	(8, 13)
5	8	(6, 8)	12	16	(9, 12)
6	5	(7, 7)	14	14	(10, 11)

Q. Digitize the circle with radius r =5 centered (2, 3).

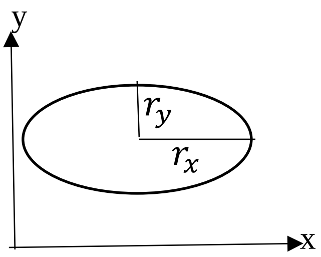
Solution:

Here,
Center = (2, 3)
Radius (r) =5
Initial point = (0, r) = (0, 5)
Initial decision parameter $p_0 = 1 - r = 1 - 5 = -4$
From mid-point circle algorithm we have;
If $p_k < 0$;
Plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$
$p_k \geq 0$;
Plot $(x_k + 1, y_k - 1)$ and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

		1st			Other pixels (in …octant)						
k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$	2nd	3rd	4th	5th	6th	7th	8th
0	-4	(1, 5)	2	10	(5,1)	(-5,1)	(-1, 5)	(-1, -5)	(-5,-1)	(5,-1)	(1, -5)
1	-1	(2, 5)	4	10	(5,2)	(-5,2)	(-2, 5)	(-2, -5)	(-5,-2)	(5,-2)	(2, -5)
2	4	(3, 4)	6	8	(4,3)	(-4,3)	(-3, 4)	(-3, -4)	(-4,-3)	(4,-3)	(3, -4)
3	3	(4, 3)	8	6	(3,4)	(-3,4)	(-4, 3)	(-4, -3)	(-3,-4)	(3,-4)	(4, -3)

Actual pixels							
1st octant	2nd octant	3rd octant	4th octant	5th octant	6th octant	7th octant	8th octant
(3, 8)	(7, 4)	(-3, 4)	(1, 8)	(1, -2)	(-3, 2)	(7, 1)	(3, -2)
(4, 8)	(7, 5)	(-3, 5)	(0, 8)	(0, -2)	(-3, 1)	(7, 1)	(4, -2)
(5, 7)	(6, 6)	(-2, 6)	(-1,7)	(-1, -1)	(-2, 0)	(6, 0)	(5, -1)
(6, 6)	(5, 7)	(-1, 7)	(-2, 6)	(-2, 0)	(-1, -1)	(5, -1)	(6, 0)

➢ **<u>Ellipse</u>**



The general equation of ellipse with semi-major axis r_x & semi-minor axis r_y centered at (x_c, y_c) is

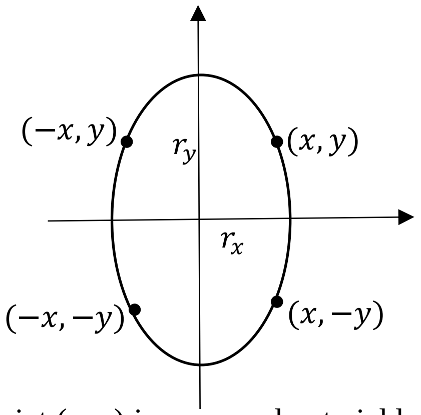
$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2$$

In polar form, the eqn. can be represented as

$$x = x_c + r_x cos\theta$$

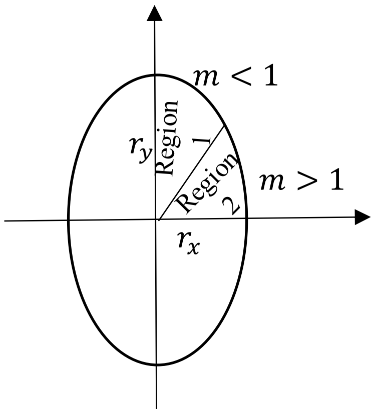
$$y = y_c + r_y sin\theta$$

<u>Symmetry of ellipse</u>



➔ Calculation of a point (x, y) in one quadrant yields the ellipse point in other three quadrant.

Ellipse processing regions



In region 1, $m < 1$

In region 2, $m > 1$

Mid-point Ellipse Algorithm:

1. Input r_x, r_y and center (x_c, y_c) and obtain the first point on ellipse centered at origin as $(x_0, y_0) = (0, r_y)$
2. Calculate the initial value of decision parameter in region 1 as
$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$
3. At each step x_k position in region 1, starting at k=0, perform the following test:
If $p1_k < 0$, the next point on ellipse centered at (0, 0) is $(x_k + 1, y_k)$ and
$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

With,

$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2,\ 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$

And continue until $2r_y^2 x \geq 2r_x^2 y$

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as
$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$
5. At each step y_k in region 2, starting at k=0, perform the following test:
If $p2_k > 0$, the next point along the ellipse centered on (0, 0) is $(x_k, y_k - 1)$ and
$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

6. Determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:
$$x = x + x_c, \quad y = y + y_c$$
8. Repeat the steps for region 1 until $2r_y^2 x \geq 2r_x^2 y$ and region 2 until $(r_x, 0)$.

Example

Q. Digitize the ellipse with $r_x = 8$, $r_y = 6$ and center at (3, 5).

Solution:

For region 1

The initial point for the ellipse at origin

$$(x_0, y_0) = (0, r_y) = (0, 6)$$

The initial decision parameter

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = 6^2 - 8^2 \times 6 + \frac{1}{4} \times 8^2 = -332$$

From midpoint algorithm, for region 1 we know,
If $p1_k < 0$ then
$x_{k+1} = x_k + 1, \quad y_{k+1} = y_k$ and $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$
If $p1_k \geq 0$ then
$x_{k+1} = x_k + 1, \quad y_{k+1} = y_k - 1$ and $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$.

k	$p1_k$	$(x_{k+1}, \ y_{k+1})$	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Now, we move out of region 1 since $2r_y^2 x_{k+1} > 2r_x^2 y_{k+1}$.

1st quadrant	2nd quadrant	3rd quadrant	4th quadrant
(4, 11)	(-4, 11)	(-4, -11)	(4, -11)
(5, 11)	(-5, 11)	(-5, -11)	(5, -11)
(6, 11)	(-6, 11)	(-6, -11)	(6, -11)
(7, 10)	(-7, 10)	(-7, -10)	(7, -10)
(8, 10)	(-8, 10)	(-8, -10)	(8, -10)
(9, 9)	(-9, 9)	(-9, -9)	(9, -9)
(10, 8)	(-10, 8)	(-10,- 8)	(10, -8)

For region 2

Initial point for region 2 $(x_0, y_0) = (7, 3)$

The initial decision parameter is

$$p2_0 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$
$$= 36 + (7 + 0.5)^2 + 64 \times 2^2 - 64 \times 36 \qquad = -23$$

From midpoint algorithm, for region 2 we know,
If $p2_k > 0$ then
$x_{k+1} = x_k, \quad y_{k+1} = y_k - 1$ and $p2_{k+1} = p2_k + 2r_x^2 y_{k+1} + r_x^2$
If $p1_k \leq 0$ then
$x_{k+1} = x_k + 1, \qquad y_{k+1} = y_k - 1$ and $p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$.

k	$p2_k$	$(x_{k+1}, \quad y_{k+1})$	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	-23	(8, 2)	576	256
1	361	(8, 1)	576	72
2	497	(8, 0)	-	-

1st quadrant	2nd quadrant	3rd quadrant	4th quadrant
(11, 7)	(-11, 7)	(-11, -7)	(11, -7)
(11, 6)	(-11, 6)	(-11,- 6)	(11, -6)
(11, 5)	(-11, 5)	(-11, -5)	(11, -5)

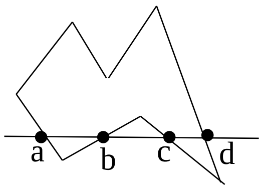
Filled Area Primitives

- Filling of polygon with solid color.
- Suppose we want to color a polygon.
- Coloring must be done only inside its boundary.
- There are two basic approaches to area filling in raster systems:
  - One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.
  - Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

Algorithms for filled area primitives

a) Scan line polygon fill algorithm
b) Boundary fill algorithm
c) Flood fill algorithm

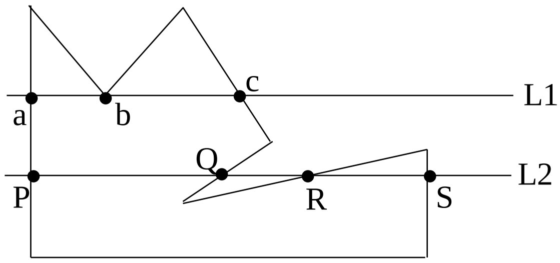
a) Scan line polygon fill algorithm



The basic scan-line algorithm is as follows:

- Find the intersections of the scan line with all edges of the polygon
- Sort the intersections points from left to right. i.e. (a, b, c, d)
- Fill in all pixels between pairs of intersections that lie interior to the polygon. i.e. (a, b) & (c, d).

Problem:



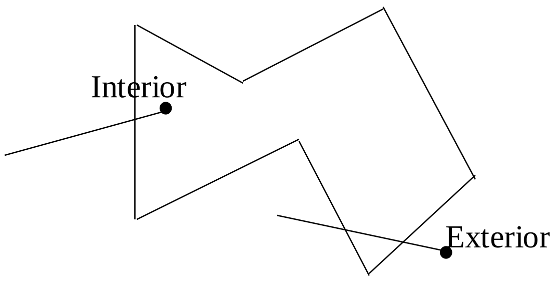
- For scan line L1 a, b & c are interior point, therefore we take pairwise points (a, b) & (b, c) and fill all the pixel between these points.
- For scan line L2, we should only take pairwise points (P, Q) & (R, S) because (Q, R) is not part of polygon.
- For scan line L1, we took the vertex 'b' twice i.e.(a, b) & (b, c) but for scan line 'L2' we did not take 'Q' twice.

  How to determine this?

Solution:

- Make a clockwise or anticlockwise traverse on the edge.
- If 'y' is monotonically increasing or decreasing and direction of 'y' changes, then we have take the vertex twice, otherwise take vertex only once.

Inside outside Test



- This test is used to identify whether a given point is inside the polygon or outside the polygon.

- ***Rule:*** To identify a point whether it is exterior or interior, draw a line from that point to outside a polygon, if this line crosses even number of edges the point is exterior otherwise it is interior.

<u>Scan-Line Fill of Curved Boundary area</u>

It requires more work than polygon filling, since intersection calculation involves nonlinear boundary. For simple curves such as circle or ellipses, performing a scan line fill is straight forward process. We only need to calculate the two scan-line intersection on opposite sides of the curve. Then simply fill the horizontal spans of pixel between the boundary points on opposite side of curve. Symmetries between quadrants are used to reduce the boundary calculation. We can fill generating pixel position along curve boundary using mid-point method

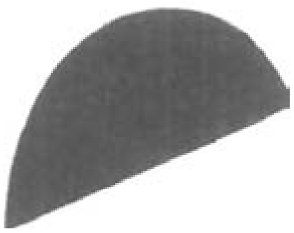
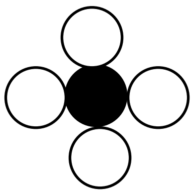
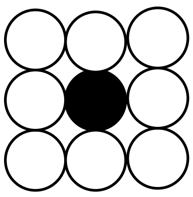


Fig: Interior fill of an elliptical arc

b) <u>Boundary fill Algorithm</u>

- It accepts an input, the co-ordinate of interior point $p(x, y)$, a fill color and a boundary color.
- Starting from point $p(x, y)$, test is performed to determine whether the neighboring pixel is already filled or boundary is reached. If not the neighboring pixels are filled with fill color and their neighbors are tested. This process is repeated till boundary is reached.
- Two mechanism is used for finding the neighboring pixel:
    - ***4-connected*** if they are adjacent horizontally and vertically.
    - ***8-connected*** if they are adjacent horizontally, vertically and diagonally.





- This algorithm is used when boundary is of single color.

Boundary fill 4-connected Algorithm

```
void Boundary_fill4(int x, int y, int b_color, int
fill_color)
{
    int value = getpixel (x, y);
    if (value! =b_color && value!=fill_color)
    {
      putpixel (x, y, fill_color);
      Boundary_fill 4 (x-1, y, b_color, fill_color);
      Boundary_fill 4 (x+1, y, b_color, fill_color);
      Boundary_fill 4 (x, y-1, b_color, fill_color);
      Boundary_fill 4 (x, y+1, b_color, fill_color);
    }
}
```

Boundary fill 8-connected Algorithm

```
void Boundary-fill8(int x,int y,int b_color, int
fill_color)
{
    int current = getpixel (x, y);
    if (current !=b_color && current!=fill_color)
    {
       putpixel (x,y,fill_color);
       Boundary_fill8(x-1, y, b_color,fill_color);
       Boundary_fill8(x+1, y, b_color, fill_color);
       Boundary_fill8(x, y-1, b_color, fill_color);
       Boundary_fill8(x, y+1, b_color, fill_color);
       Boundary_fill8(x-1, y-1, b_color,fill_color);
       Boundary_fill8(x-1, y+1, b_color,fill_color);
       Boundary_fill8(x+1, y-1, b_color,fill_color);
       Boundary_fill8(x+1, y+1, b_color,fill_color);
     }
}
```

Note: Recursive boundary fill algorithm doesn't fill regions correctly if some interior pixels are already displayed in fill color. To avoid this problem, initially set all the color of interior pixel to a specified color.

c) ***Flood Fill Algorithm***

- This algorithm is used when boundary is of different color.
- We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with desired fill-color.

Algorithm (for 4-connected)

```
void flood_fill4(int x, int y, int fill_color, int old_color)
{
    int current = getpixel (x,y);
    if (current==old_color)
   {
      putpixel (x,y,fill_color);
      flood_fill4(x-1, y, fill_color, old_color);
      flood_fill4(x+1, y, fill_color, old_color);
      flood_fill4(x, y-1, fill_color, old_color);
      flood_fill4(x, y+1, fill_color, old_color);
    }
}
```

- Similarly flood fill for 8 connected can be also defined.

Difference between Boundary and Flood fill Algorithm

Boundary fill Algorithm	Flood fill Algorithm
Area filling is started inside a point within a boundary region and fill the region with in the specified color until it reaches the boundary.	Area filling is started from a point and it replaces the old color with the new color.
It is used in interactive packages where we can specify the region boundary.	It is used when we cannot specify the region boundary.
It is less time consuming.	It consumes more time.
It searches for boundary.	It searches for old color.

References

- **Donald Hearne and M.Pauline Baker**, “Computer Graphics, C Versions.” Prentice Hall